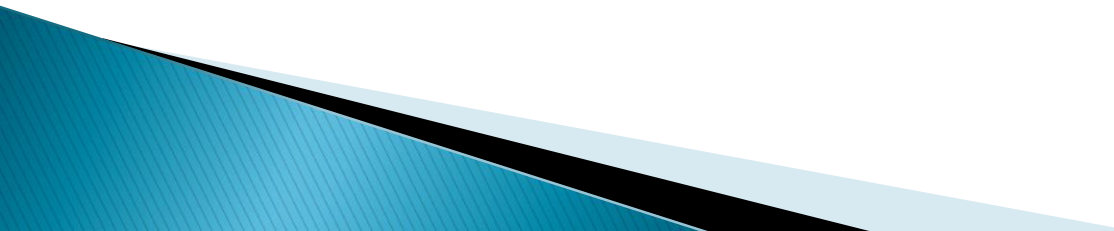




Fuzzing

# What is fuzzing?

- ▶ Sending input to an application, that the application is not expecting, to try to make it crash
    - Long strings of characters that might overflow a memory buffer
    - Weird characters that the application isn't expecting
    - Malformed input files
    - Etc
- 

# Why fuzz?

- ▶ I wanted to learn about fuzzing, to assist with my learning about exploit development
- ▶ Exploitable code is commonly found via fuzzing
  - Especially if you don't have access to the source code
- ▶ We will be focusing on binary fuzzing
  - Web app fuzzing is another topic (although very common)

# Why fuzz?

- ▶ Some deep-in-code bugs are being found, that have been present for many years
  - I was curious about how they were being found
  - And why they weren't found sooner
- ▶ Basically, the only way to find these deep-in-code bugs is thru fuzzing
  - It is all but impossible to manually test all of the input combinations

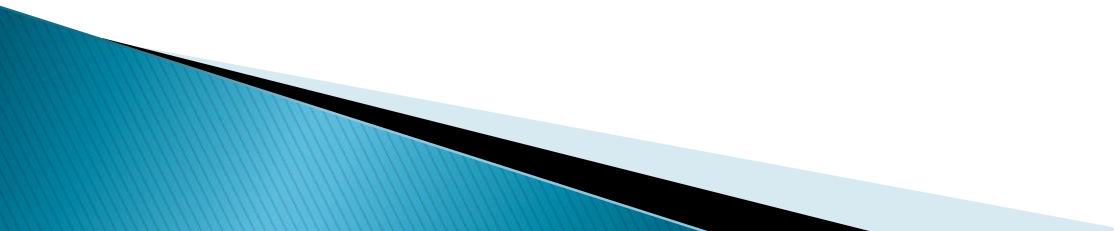
# Why fuzz?

- ▶ Many security testers and researchers use fuzzing to find these deep-in-code vulnerabilities
  - Many software companies fuzz their products before they are released to market
  - 3<sup>RD</sup> parties also fuzz companies products, to catch remaining bugs
    - Google Project Zero is famous
    - Other organizations as well
  - Many open-source projects are also being fuzzed by security researchers, to find and fix bugs in them

# Disclaimer

- ▶ Note that this is a very basic primer of fuzzing
  - You can go much deeper and further, expanding code coverage, etc
  - But my goal right now was just to learn the basics

# Malformed input file fuzzing

- ▶ Using the Zzuf tool, make random small changes to a file. The resulting file can then be fed into an application, to try to make it crash.
  - ▶ Image files
  - ▶ Documents, Pdf files
  - ▶ Etc
- 

# Malformed input file fuzzing

- ▶ 1) Modify some pictures a bunch of times
  - cat.\* is cat pictures, not the linux cat command

```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ for i in {1000..3000}; do for f in cat.*; do zzuf -r 0.01 -s $i < "$f" > "$i-$f"; done; done
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$
```

- ▶ 2) Feed the modified pictures into an image converter, save error messages to log file
  - Image Magick, any others

```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ LC_ALL=C; LANG=C; for f in *-cat.*; do timeout 3 convert -resize 2 "$f" /tmp/test.png; echo $f; done &> fuzzing.log
```



# Malformed input file fuzzing

- ▶ 3) Look in the log file for segmentation faults

```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ tail fuzzing.log
convert-im6.q16: Image width is zero in IHDR `/tmp/test.png' @ warning/png.c/MagickPNGWarningHandler/1668.
convert-im6.q16: Image height is zero in IHDR `/tmp/test.png' @ warning/png.c/MagickPNGWarningHandler/1668.
convert-im6.q16: Invalid IHDR data `/tmp/test.png' @ error/png.c/MagickPNGErrorHandler/1642.
1630-cat.jpeg
convert-im6.q16: no images defined `/tmp/test.png' @ error/convert.c/ConvertImageCommand/3229.
1630-cat.png
1630-cat.tga
convert-im6.q16: improper image header `1630-cat.xwd' @ error/xwd.c/ReadXWDImage/316.
convert-im6.q16: no images defined `/tmp/test.png' @ error/convert.c/ConvertImageCommand/3229.
1630-cat.xwd
```

```
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$ cat fuzzing.log | grep "seg"
lab@lab-VirtualBox:~/fuzzingDemo/inputFileFuzzing$
```

# Malformed text entry fuzzing

- ▶ Using AFL (American Fuzzy Lop)
  - Also the name of a rabbit breed, if you search for help for the tool
- ▶ It takes text input via input files, and then morphs it to try to break things
- ▶ It also tries to spider out through the code to hit all paths and get good coverage

# Malformed text entry fuzzing

- ▶ Warnings:
- ▶ It requires at least 1 input that does not cause an error
- ▶ And if an input causes an error, it will not try to morph that input further
  - bad json file = won't try to further morph that json

# Malformed text entry fuzzing

- ▶ Run AFL on the binary
  - Specifying the directory that contains the input files
  - And the directory that AFL should put the output files to

```
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master$ afl-fuzz -i in -o out ./fuzzgoat @@
```

# Malformed text entry fuzzing

- ▶ AFL will show its progress
  - Note the total crashes counter in red on the right

```
american fuzzy lop ++4.00c {default} (./fuzzgoat) [fast]sults
┌───────────┴───────────┐
┌───────────┐┌───────────┐┌───────────┐
│ process timing │ │ overall results │ │
│   run time   : 0 days, 0 hrs, 0 min, 33 sec │ │ cycles done   : 0 │ │
│ last new find : 0 days, 0 hrs, 0 min, 0 sec │ │ corpus count  : 94 │ │
│ last saved crash : 0 days, 0 hrs, 0 min, 8 sec │ │ saved crashes : 5 │ │
│ last saved hang  : none seen yet │ │ saved hangs   : 0 │ │
├───────────┴───────────┤├───────────┐├───────────┐
│ cycle progress │ │ map coverage │ │ findings in depth │ │
│ now processing : 0.0 (0.0%) │ │ map density   : 0.00% / 0.00% │ │
│ runs timed out : 0 (0.00%) │ │ count coverage : 1.84 bits/tuple │ │
├───────────┴───────────┤├───────────┐├───────────┐
│ stage progress │ │ favored items : 1 (1.06%) │ │
│ now trying     : havoc │ │ new edges on  : 56 (59.57%) │ │
│ stage execs    : 10.6k/32.8k (32.45%) │ │ total crashes : 5 (5 saved) │ │
│ total execs    : 11.4k │ │ total tmouts  : 17 (5 saved) │ │
│ exec speed     : 310.7/sec │ │
├───────────┴───────────┤├───────────┐├───────────┐
│ fuzzing strategy yields │ │ item geometry │ │
│ bit flips      : disabled (default, enable with -D) │ │ levels       : 2 │ │
│ byte flips     : disabled (default, enable with -D) │ │ pending      : 94 │ │
│ arithmetics    : disabled (default, enable with -D) │ │ pend fav     : 1 │ │
│ known ints     : disabled (default, enable with -D) │ │ own finds    : 93 │ │
│ dictionary     : n/a │ │ imported     : 0 │ │
│ havoc/splice   : 0/0, 0/0 │ │ stability    : 100.00% │ │
│ py/custom/rq   : unused, unused, unused, unused │ │
│ trim/eff       : 0.00%/1, disabled │ │
├───────────┴───────────┤├───────────┐├───────────┐
│                                     │ │ [cpu:200%] │ │
└───────────┴───────────┘└───────────┘└───────────┘
```



# Malformed text entry fuzzing

- ▶ AFL will put the crashes in files in the out/default/crashes directory
  - Each file will contain an input that caused a crash

```
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default$ cd crashes/  
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ ls  
id:000000,sig:11,src:000000,time:3629,execs:1070,op:havoc,rep:4  
id:000001,sig:11,src:000000,time:4948,execs:1354,op:havoc,rep:2  
id:000002,sig:06,src:000000,time:8135,execs:2493,op:havoc,rep:8  
id:000003,sig:11,src:000000,time:14503,execs:4955,op:havoc,rep:2  
id:000004,sig:06,src:000000,time:25527,execs:8659,op:havoc,rep:4  
README.txt
```

```
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ cat id  
\:000000\,sig\:11\,src\:000000\,time\:3629\,execs\:1070\,op\:havoc\,rep\:4  
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ cat  
\:000001\,sig\:11\,src\:000000\,time\:4948\,execs\:1354\,op\:havoc\,rep\:2  
{"":12}  
lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes$ cat id  
\:000003\,sig\:11\,src\:000000\,time\:14503\,execs\:4955\,op\:havoc\,rep\:2  
{"":"'"}lab@lab-VirtualBox:~/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes  
$
```

# Gui fuzzing

- ▶ Not straight-forward like sending a command line argument or input file to an application
- ▶ Instead have to simulate mouse clicks and keyboard strokes
  - To random places
  - Or to specific spots
- ▶ No standard fuzzing tools available, since it needs to be so customized

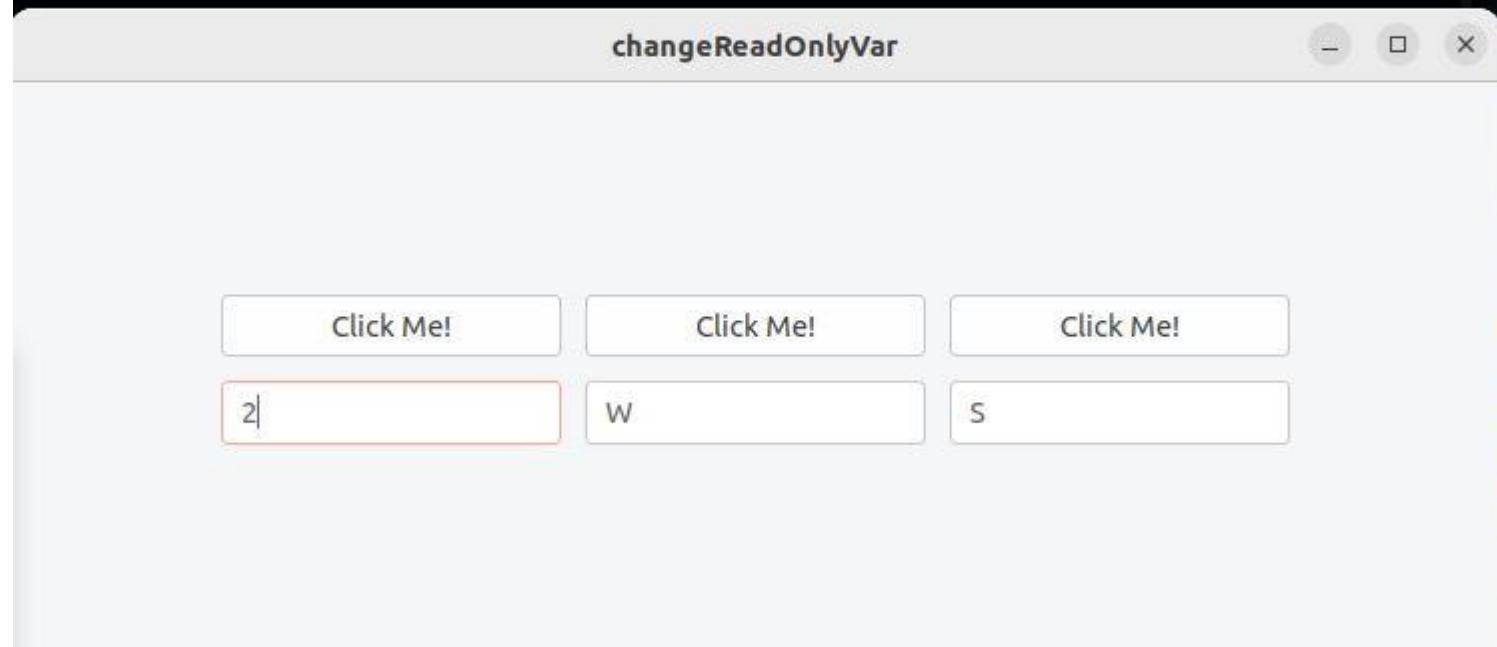
# Gui fuzzing – Demo

- ▶ I made a simple python-based fuzzer
  - That clicks at random places within an application's window
  - And tabs through the application's gui elements, and sends random characters
- ▶ Poc vulnerable c program
  - That does a segfault when the left button is clicked 3x
  - Or when “3” is entered into the middle textbox



# Gui fuzzing – Demo

```
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$ ./changeReadOnlyVar  
Gtk-Message: 15:35:52.515: Failed to load module "canberra-gtk-module"  
Entry contents: W  
button clicked 1 times  
□
```

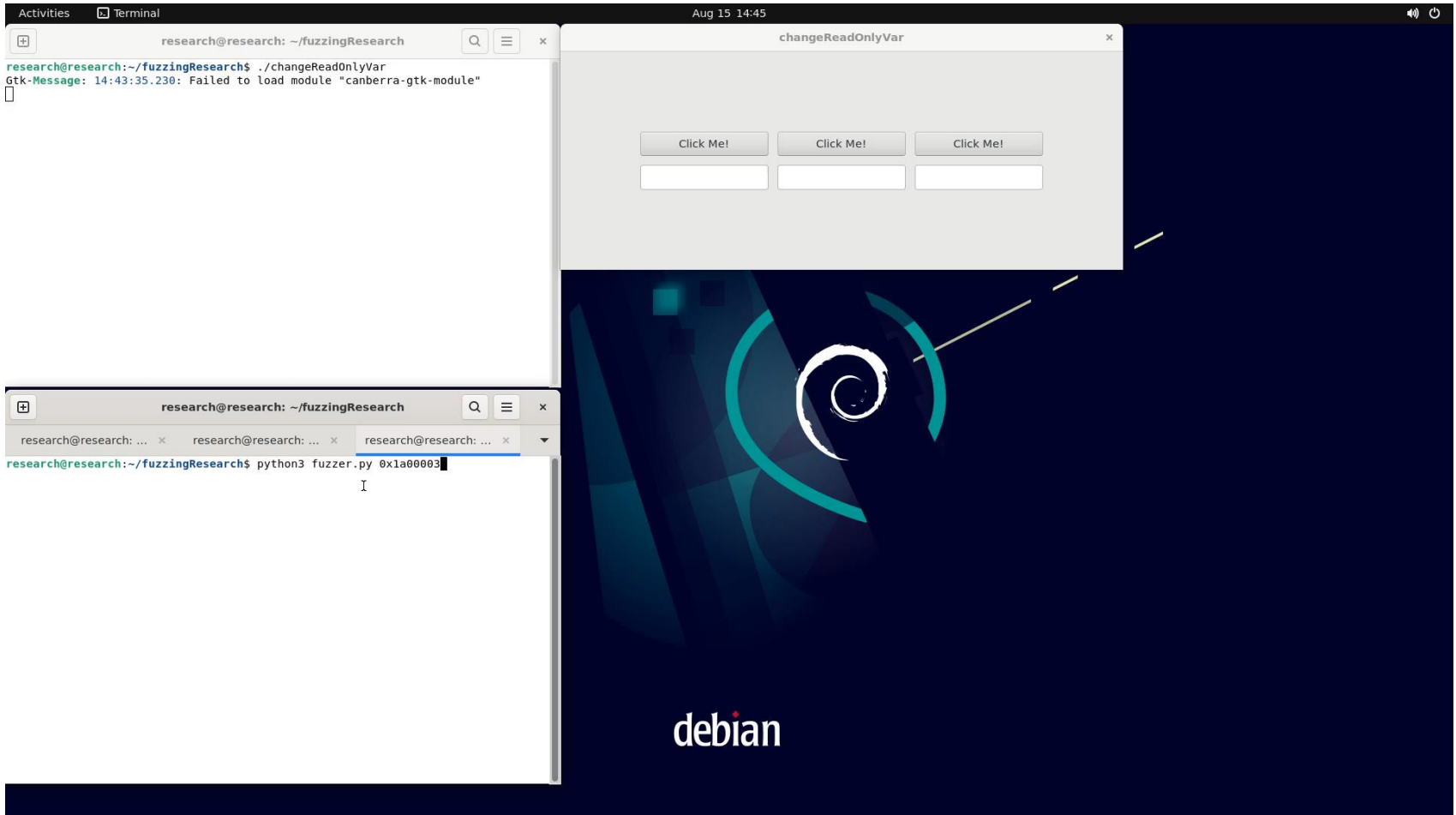


# Gui fuzzing – Demo

```
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$ python3 fuzzer.py 0x1000003
```

```
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$ ./changeReadOnlyVar
Gtk-Message: 15:20:03.503: Failed to load module "canberra-gtk-module"
button clicked 1 times
Entry contents: a
button clicked 2 times
button clicked 3 times
Segmentation fault (core dumped)
lab@lab-VirtualBox:~/fuzzingDemo/guiFuzzing$
```

# Gui fuzzing – Demo

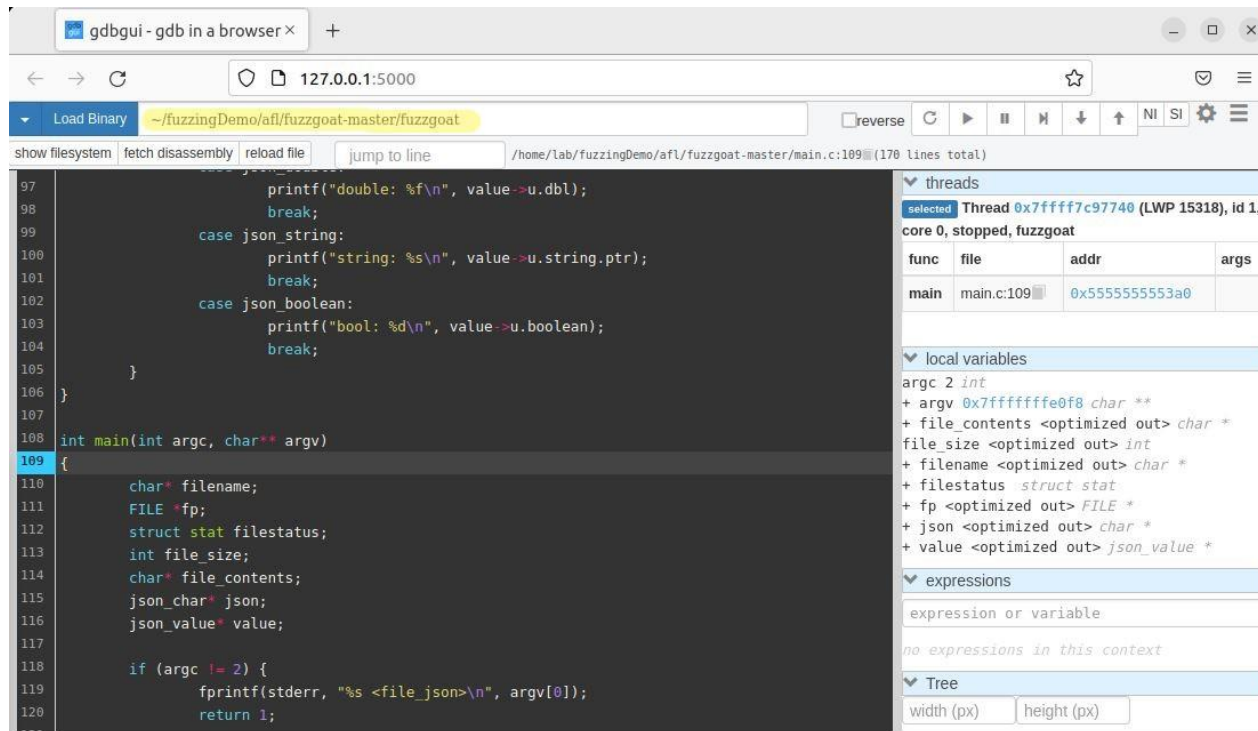


# Debugging

- ▶ Not every crash / segfault is exploitable
  - There could be runtime or memory protections that prevent you from taking further control
- ▶ The next step is to run the application through a debugger
  - To see what code section the application crashes in
  - To then see how you may be able to further exploit the code
  - Use the malformed input that caused the crash

# Debugging

## ► Load the target binary



```
Starting program: /home/lab/fuzzingDemo/afl/fuzzgoat-master/fuzzgoat validObject
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

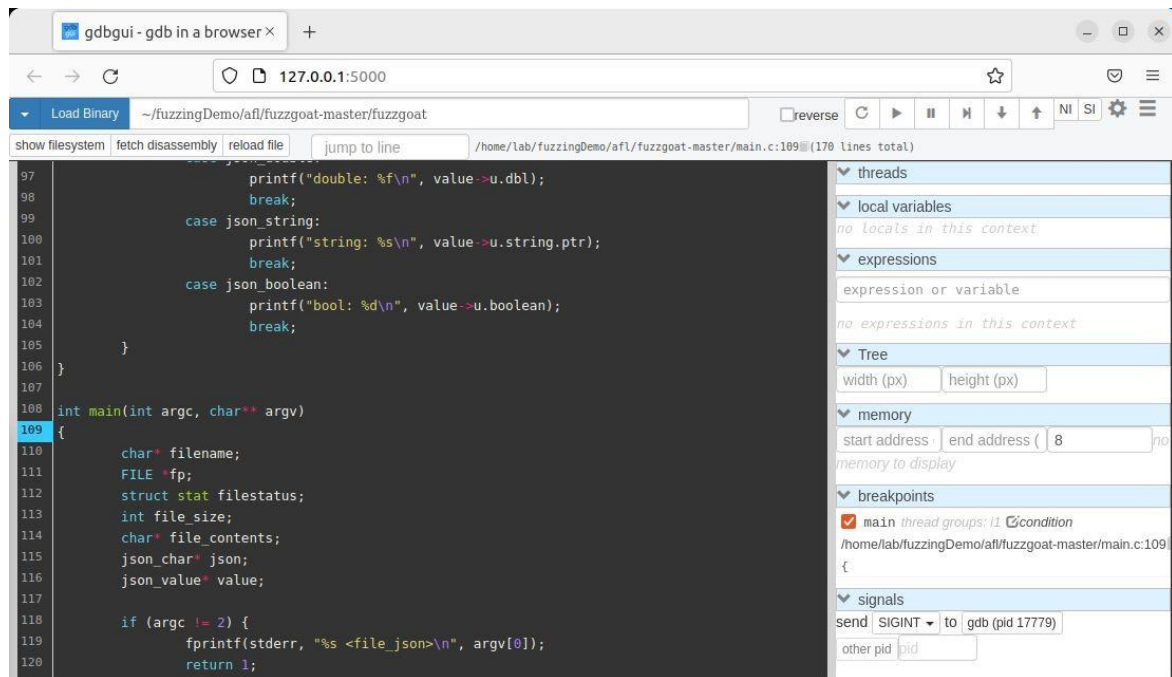
```
Breakpoint 1, main (argc=2, argv=0x7ffff7c97740) at main.c:109
109 {
(gdb)
```

```
gdbgui output (read-only)
Copy/Paste available in all terminals with ctrl+shift+c, ctrl+shift+v
Started new gdb process, pid 15310
Selected thread is running.
```

```
Program output -- Programs being debugged are connected to this terminal. You can read output and send input to the program from here.
```

# Debugging

- ▶ Run the target binary, with the arguments that cause the crash



tion resources online at:  
<<http://www.gnu.org/software/gdb/Documentation/>>.

For help, type "help".  
Type "apropos word" to search for comma  
nds related to "word".

New UI allocated  
(gdb) run /home/lab/fuzzingDemo/afl/fuz  
zgoat-master/out/default/crashes/id:000  
001,sig:11,src:000000,time:4948,execs:1  
354,op:havoc,rep:2

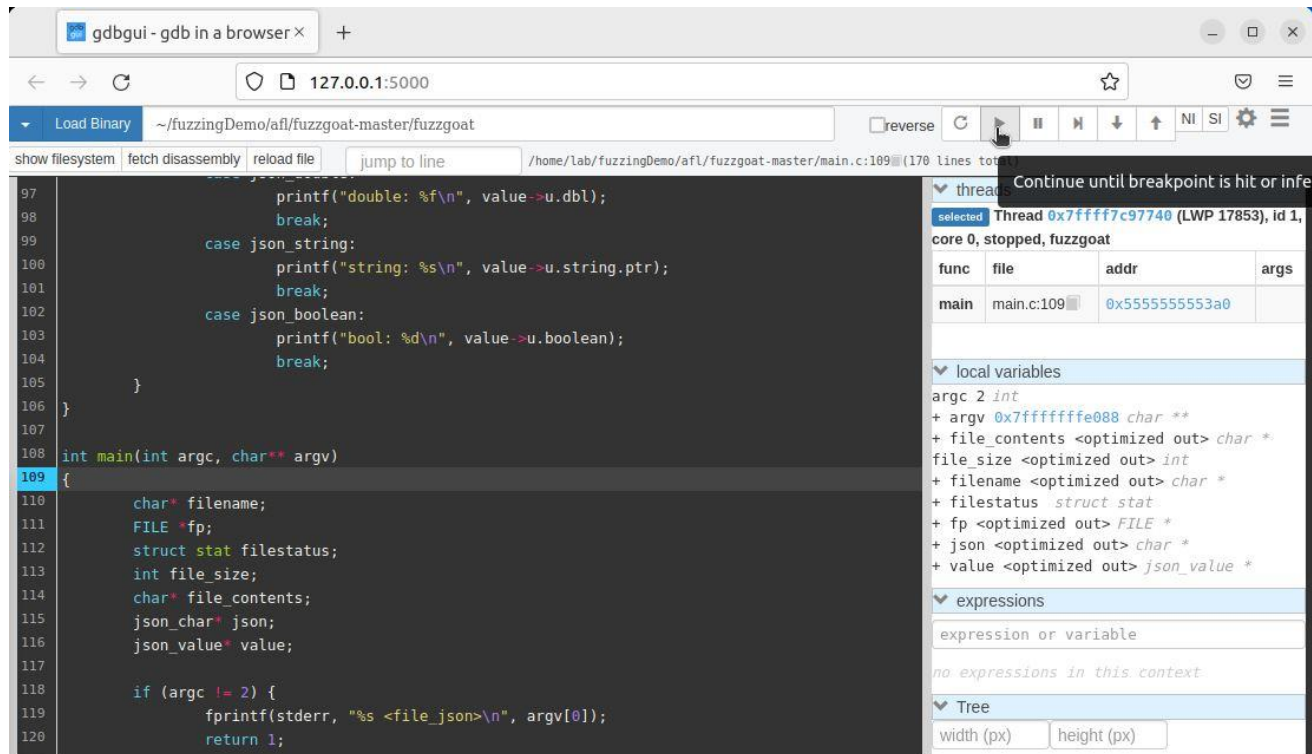
gdbgui output (read-only)  
Copy/Paste available in all terminals w  
ith ctrl+shift+c, ctrl+shift+v  
Started new gdb process, pid 17779

Program output -- Programs being debugg  
ed are connected to this terminal. You  
can read output and send input to the p  
rogram from here.



# Debugging

- ▶ Click the play button, and let it run



```
/fuzzingDemo/afl/fuzzgoat-master/out/default/crashes/id:000001,sig:11,src:000000,time:4948,execs:1354,op:havoc,rep:2
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

```
Breakpoint 1, main (argc=2, argv=0x7ffff7c97740) at main.c:109
109 {
(gdb) █
```

```
gdbgui output (read-only)
Copy/Paste available in all terminals with
ctrl+shift+c, ctrl+shift+v
Started new gdb process, pid 17779
Selected thread is running.
```

```
Program output -- Programs being debugged
are connected to this terminal. You
can read output and send input to the
program from here.
```

# Debugging

- ▶ When it crashes, it will stop at the line of code that caused the crash

The screenshot displays the gdbgui web interface in a browser. The address bar shows the URL `127.0.0.1:5000`. The interface is divided into several panels:

- Source Code Panel:** Shows the source code of `fuzzgoat.c`. The current line of execution is highlighted at line 224: `switch (value->type)`. The code includes a `while` loop and a `switch` statement.
- Threads Panel:** Lists the threads running in the process. The selected thread is `Thread 0x7ffff7c97740 (LWP 17853), id 1, core 0, stopped, fuzzgoat`.
- Local Variables Panel:** Displays the current state of local variables, including `cur_value`, `settings`, `settings@entry`, `value`, and `value@entry`.
- Expressions Panel:** Shows the current state of expressions, with a message indicating `no expressions in this context`.
- Tree Panel:** Displays the current state of the tree, with a message indicating `no expressions in this context`.

The bottom panel shows the output of the program, which includes a segmentation fault message: `Program received signal SIGSEGV, Segmentation fault.` The output also shows the backtrace, indicating the fault occurred in `json_value_free_ex` at line 224.



# Debugging

- ▶ You can then investigate what the code did to cause the crash
  - But that is another massive topic for another presentation

# Questions

# ?

